

Inhaltsverzeichnis

2.1 Der Objektbegriff 24

2.2 Einfache Ereignisprozeduren zum Standard-Ereignis 29

2.3 Einfache Ereignisprozeduren zu Nicht-Standard-Ereignissen 35

Im vorigen Kapitel wurde geschildert, wie Delphi gestartet wird, wie die Starteigenschaften der grundlegenden Arbeitsfläche, des Formulars, voreingestellt werden, wie man zweckmäßig speichert. Weiter wurde erläutert, wie die so genannte *Laufzeit* gestartet wird, in der die vorbereitete Arbeitsfläche dann tatsächlich erscheint und in der man kontrollieren kann, ob die vorbereitete Erscheinungsform des Formulars wie gewünscht zu sehen ist.

Daran anschließend folgten erste Anleitungen, wie wichtige *Bedienelemente* auf dem Formular platziert und ihrerseits für ihre wesentlichen Starteigenschaften voreingestellt werden können: *Button* (Schaltfläche), *Label* (Beschriftungsfeld), *Edit* (Texteingabefeld, Textfenster), *Checkbox* (Ja-Nein-Option), *Scrollbar* (Schieberegler) sowie *Radiobuttons* in *GroupBoxen* (gruppierte Auswahl).

Bis hierher ließ sich alles schön einfach, „rein handwerklich“, schildern. Man nehme, man tue. Bis auf den Umgang mit einigen englischen Vokabeln, deren Kenntnis für die Suche nach den Eigenschaften und deren Bedeutung im *Objektinspektor* hilfreich ist, wurde unser Denkvermögen noch nicht auf harte Proben gestellt.

Doch wenn wir inhaltlich verstehen wollen, was tatsächlich bei der Delphi-Programmierung vor sich geht, wenn wir eigenen Anteil und die Leistungen des Delphi-Systems in ihrem Zusammenhang und Wechselspiel einordnen möchten, müssen wir uns näher mit dem Begriff des *Objektes* befassen.

Denn schließlich werden wir in Wirklichkeit nichts Anderes als *OOP* betreiben – *Objektorientierte Programmierung*. Nur fünf Kapitel werden wir benötigen, und dann werden wir wissen, was wir tun.

Wir gehen in kleinen Schritten vor. Befassen wir uns zuerst mit den Begriffen *Objekt*, *Ereignis* und *Ereignisprozedur*.

2.1 Der Objektbegriff

2.1.1 Datenobjekte

Ein *Datenobjekt* besteht zuallererst aus einem *geschützten Datenkern*. Man sagt auch, die dort enthaltenen Daten sind gekapselt.

Wie der Datenkern im Einzelnen aufgebaut ist, das weiß nur derjenige Programmierer, der dieses Datenobjekt irgendwann einmal vorbereitet hat.

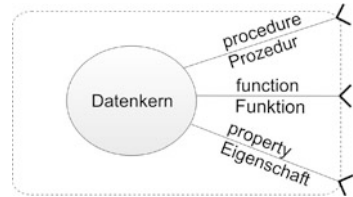
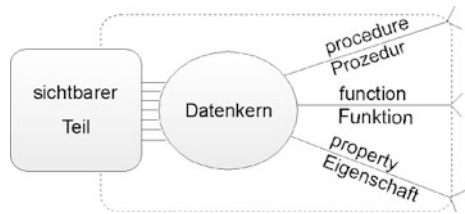
Ist ein Objekt vorbereitet (und beispielsweise in eine Objektsammlung aufgenommen worden), so können viele andere Programmierer damit arbeiten. Wir wollen diese zum Unterschied zu dem Vorbereiter des Objekts als *Nutz-Programmierer* bezeichnen.

Kein Nutz-Programmierer kann aber unmittelbar auf die Bestandteile des Datenkerns zugreifen.

Hätte ein Objekt nur den geschützten Datenkern, so wäre es offensichtlich sinnlos. Niemand, kein Nutz-Programmierer, könnte mit den Daten arbeiten. Deshalb enthält jedes Objekt zusätzlich gewisse Mechanismen, mit deren Hilfe jeder Nutz-Programmierer mit dem Datenkern umgehen kann.

In Abb. 2.1 sind die drei Mechanismen eingezeichnet, die zu jedem Delphi-Objekt gehören können:

- Eine *property* (Eigenschaft) dient dem schnellen aktiven und passiven Zugriff auf einzelne Bestandteile des Datenkerns. Verwendet ein Nutz-Programmierer eine *property aktiv*, dann *verändert* er mit ihrer Hilfe einen einzelnen Wert im Datenkern. *Passive Verwendung* dagegen *informiert* über die aktuelle Situation im Datenkern, ohne darin zu ändern.

Abb. 2.1 Objekte in Delphi**Abb. 2.2** Visuelles Objekt in Delphi

Wenn z. B. im Datenkern die Uhrzeit mit Stunde und Minute verwaltet wird, könnten über zwei properties sowohl die Einzelwerte abgefragt als auch verändert werden. Properties werden verwendet.

- Eine *function* (Funktion) liefert stets *einen einzelnen Wert*, den Ergebniswert, der in Zusammenhang mit dem Inhalt des Datenkerns steht.

Wird z. B. im Datenkern ein Text verwaltet, so könnte eine Funktion die aktuelle Anzahl der Zeichen liefern. Funktionen werden im Allgemeinen *passiv* verwendet. Funktionen können aber auch im Datenkern verändern. Ihr Ergebniswert muss dann nicht immer verarbeitet werden (siehe die Funktion `Add`, Abschn. 13.3.6). Dann wird die Funktion ausnahmsweise *aufgerufen*.

- Eine *procedure* (Prozedur) kann eine Vielfalt an *Wirkungen* haben: Sie kann im Datenkern verändern, sie kann viele Werte liefern, kurz, sie hat eine bestimmte Wirkung. Prozeduren werden *aufgerufen*.

2.1.2 Visuelle Objekte

Delphi enthält eine beachtliche Sammlung von vorbereiteten Objekten. Im ersten Kapitel haben wir davon schon – unbewusst – Gebrauch gemacht. Wir haben nämlich besondere Objekte, die als *visuelle Objekte* bezeichnet werden, im Entwurf vorbereitet und zur Laufzeit erzeugen lassen.

Visuelle Objekte (siehe Abb. 2.2) besitzen zusätzlich einen *sichtbaren Teil*, den wir als *Bedienelement* auf dem Formular erleben. Auch das Formular selbst ist nichts anderes als der sichtbare Teil eines entsprechenden visuellen Objektes. Alle Werte, die zum sichtbaren Teil gehören, sind im Datenkern des Objekts abgespeichert.

Dazu gehören zum Beispiel Position, Größe und Farbe jedes Bedienelements, Farbe und Start-Status des Formulars, Beschriftung des Labels oder Inhalt des Textfensters, dazu gehören die Belegung bei der Checkbox oder dem Radiobutton, die Position des Reglers bei einer Scrollbar usw.

Wird folglich *im Datenkern eines visuellen Objekts* irgendein Wert verändert, der mit dem sichtbaren Teil in Beziehung steht, sehen wir sofort eine *Änderung an dem Bedienelement*.

Und schon lässt sich auch erklären, was wir eigentlich taten, als wir unter Nutzung der Rubrik EIGENSCHAFTEN des Objektinspektors in den Abschn. 1.2.1 und 1.4 jeweils die Startsituation für Formular und Bedienelemente festlegten:

Abb. 2.3 zeigt es: Einige der *properties*, die in den Datenkern führen, können von uns bereits in der Entwurfsphase mit Hilfe des *Objektinspektors* voreingestellt werden. Zu Beginn der Laufzeit, wenn dann das visuelle Objekt tatsächlich erzeugt wird, transportiert Delphi die voreingestellten Werte über die *properties* in den Datenkern.

Dort sorgen sie sofort für entsprechende Darstellung des Bedienelements, also des sichtbaren Teils des visuellen Objekts: Das Label erhält die voreingestellte Beschriftung, die Checkbox die voreingestellte Belegung, die Scrollbar die voreingestellte Position, das Formular die voreingestellte Farbe und so weiter.

Nicht alle, sondern nur die wichtigsten Bestandteile des Datenkerns sind vorab durch den Objektinspektor mit Hilfe von *properties* einstellbar.

Im Abschn. 4.7.2 werden wir erfahren, wie weitere *properties* gefunden und genutzt werden können.

Abb. 2.3 Voreinstellung der Starteigenschaften im Objektinspektor

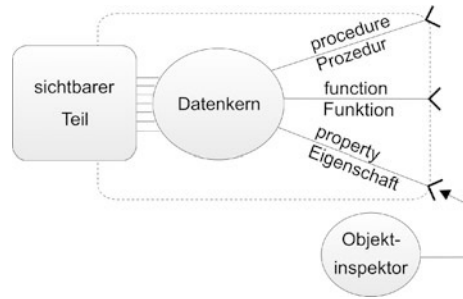
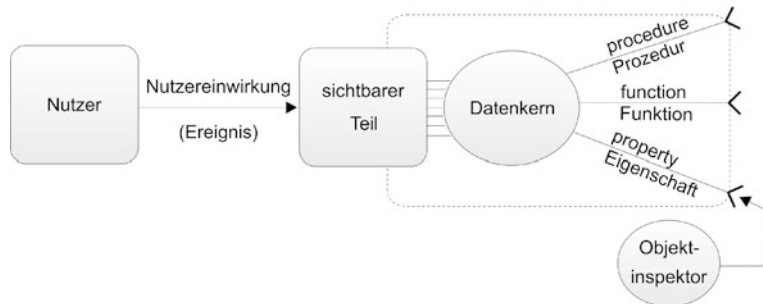


Abb. 2.4 Nutzereinwirkung ist ein Ereignis



So weit, so gut. Doch was passiert zur Laufzeit, wenn das Formular mit den darauf platzierten Bedienelementen hergestellt ist? Dann gibt es *die Nutzerin* oder *den Nutzer* (der Einfachheit halber soll generell nur kurz vom *Nutzer* gesprochen werden).

Natürlich kann ein Nutzer, er wird es sogar, auf die *Bedienelemente einwirken*: Er wird auf den Button klicken, in das Textfenster eintragen, in der Checkbox den Haken setzen oder wegnehmen, in der Scrollbar den Regler verschieben usw. Das ist sein gutes Recht, das soll er auch. Schließlich wird die Benutzeroberfläche nicht zum begeisternden Ansehen vorbereitet und hergestellt, sondern zum *Entgegennehmen von Nutzereinwirkungen*.

In der Abb. 2.4 haben wir folglich das *Schema des visuellen Delphi-Objekts* erweitert durch *Nutzer* und dessen Möglichkeit, auf ein Bedienelement einzuwirken.

Jede Nutzereinwirkung wird dabei als *Ereignis* bezeichnet.

Damit können wir uns das Wechselspiel vorstellen:

- Wird im *Datenkern* ein Bestandteil, der zur sichtbaren Komponente gehört, geändert, ändert sich automatisch der sichtbare Teil, das *Bedienelement*.
- Erfolgt andererseits eine ändernde *Nutzereinwirkung* auf ein Bedienelement, ändert sich nicht nur dieses, sondern sofort wird auch im *Datenkern* diese Änderung registriert.

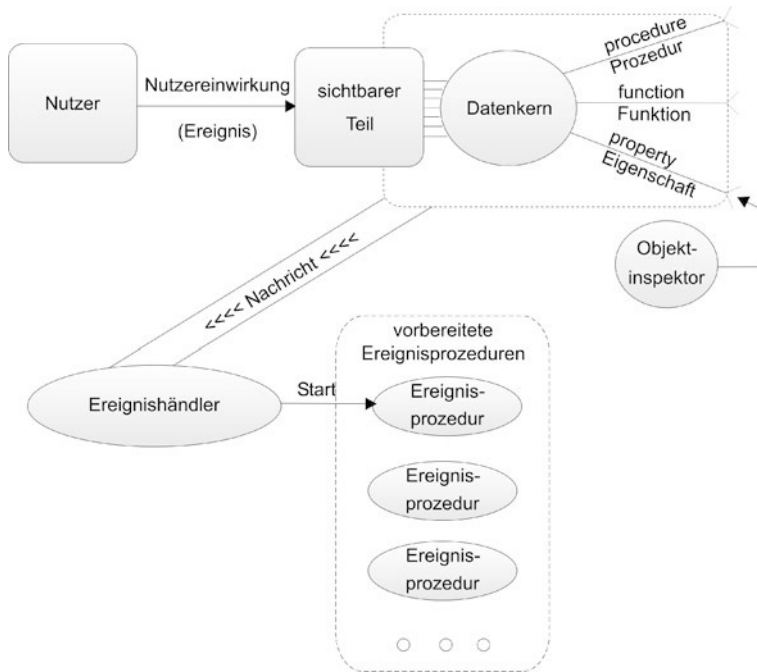


Abb. 2.5 Ereignis, Ereignishandler und Ereignisprozeduren

Doch nicht jede Nutzereinwirkung ändert etwas – so kann der Nutzer auf einen Button klicken, die Maus über dem Formular bewegen, auf ein Label klicken usw. Dabei wird nichts Sichtbares passieren, obwohl zweifelsohne eine Aktivität des Nutzers stattgefunden hat.

Wir brauchen einen übergeordneten Begriff, und das ist eben das *Ereignis*.

- Jede *Nutzerhandlung* am Formular oder an einem darauf befindlichen Bedienelement ist ein *Ereignis*.
- Einige dieser Ereignisse können zusätzlich noch Änderungen an den Bedienelementen bewirken. Finden solche Änderungen statt werden sie sofort im Datenkern des jeweiligen visuellen Objekts berücksichtigt.

2.1.3 Ereignisbehandlung

Klickt ein Nutzer zum Beispiel auf einen *Button*, dann will er, dass etwas *passiert*, dass eine *Reaktion* eintritt. Das ist sein gutes Recht – und warum gäbe es sonst überhaupt dieses Bedienelement *Button*?

Die Abb. 2.5 versucht darzustellen, wie es überhaupt dazu kommen kann, dass auf ein *vom Nutzer ausgelöstes Ereignis* schließlich eine *Reaktion* erfolgt.

- Wenn wir ein Bedienelement in der Entwurfsphase auf dem Formular platzieren, bereiten wir damit ein *visuelles Objekt* vor. Das Formular selbst wird dabei auch ein visuelles Objekt.
- Gleichzeitig wird *jedes vorbereitete visuelle Objekt* durch Delphi beim *Ereignishändler* angemeldet – wir müssen das nicht tun und merken auch nichts davon. Die Anmeldung erfolgt automatisch.
- Zur *Laufzeit*, wenn die *Benutzeroberfläche* (Formular mit darauf befindlichen Bedienelementen) tatsächlich hergestellt ist, wartet der Ereignishändler auf *Nachrichten* von den visuellen Objekten.
- Löst ein Nutzer durch eine Bedienhandlung ein *Ereignis an einem visuellen Objekt* aus, erhält der *Ereignishändler* eine entsprechende *Nachricht*.
- Der *Ereignishändler* prüft, ob es zu diesem Ereignis an diesem visuellen Objekt eine *vorbereitete Ereignisprozedur* gibt.
- Wenn es eine vorbereitete Ereignisprozedur für dieses bestimmte Ereignis an diesem Objekt gibt, wird sie *gestartet*.

Daraus folgt: Wollen wir, dass eine *Reaktion auf eine bestimmte Nutzerhandlung* an dem sichtbaren Teil eines bestimmten visuellen Objekts eintritt, müssen wir dafür sorgen, dass dafür eine *Ereignisprozedur* existiert.

Das bedeutet, dass wir uns nun mit der Frage beschäftigen müssen, wie wir Ereignisprozeduren herstellen können.

2.2 Einfache Ereignisprozeduren zum Standard-Ereignis

Jede Ereignisprozedur besteht aus dem Rahmen und dem Inhalt. Um den Rahmen brauchen wir uns bei Delphi nicht zu kümmern, der wird uns stets quasi „geschenkt“.

Zu jedem visuellen Objekt gibt es *zwei Arten von Ereignissen*: Das *Standard-Ereignis* und viele *andere Ereignisse*. Das sind dann die *Nicht-Standard-Ereignisse*.

Das *Standard-Ereignis* ist dasjenige, das in der Regel oder am häufigsten der Ausgangspunkt für eine Reaktion sein wird. Beim Bedienelement *Button* wäre es ziemlich verblüffend, wenn nicht der *Klick* mit der linken (Haupt-)Maustaste als Standard-Ereignis betrachtet würde.

Beginnen wir. Zuerst werden wir eine Reaktion auf das Standard-Ereignis an einem visuellen Objekt in einer Ereignisprozedur programmieren.

Den *Rahmen für Ereignisprozeduren zum Standard-Ereignis* erhalten wir in diesem Fall ganz einfach: Durch Doppelklick im Entwurf auf den sichtbaren Teil des visuellen Objekts.

Sehen wir uns das im Einzelnen für unsere bisher verwendeten visuellen Objekte an, d. h. für die sechs Bedienelemente und das Formular.

2.2.1 Button

Wichtiger Hinweis: Wer das folgende Delphi-Projekt nicht selbst entwickeln möchte, kann auf der Seite www.w-g-m.de/delphi.htm unter *Dateien für Kapitel 2* die Datei `DKap02.zip` herunterladen, die die Projektdatei `proj_221.dpr` enthält.

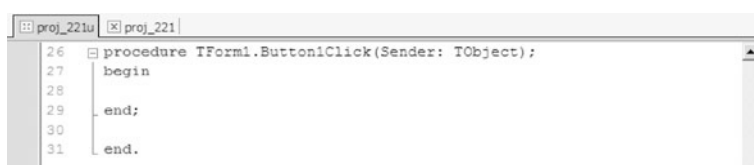
Beginnen wir: Auf einem kleinen *Formular* (es muss beim Start diesmal nicht den ganzen Bildschirm ausfüllen) platzieren wir ein *Bedienelement Button* und ändern mit der Eigenschaft `Caption` die Beschriftung auf *Start*:



Delphi schlägt für dieses Bedienelement den Namen `Button1` vor. Nun wird dazu die folgende Aufgabe formuliert: Bei einfachem Klick auf diesen Button soll der Nutzer eine Mitteilung „Der Button wurde geklickt“ erhalten. Mehr erstmal nicht.

Der *einfache Klick* ist das *Standard-Ereignis* für das *Bedienelement Button*.

Folglich beschaffen wir uns den Rahmen für die Ereignisprozedur, indem wir mit der linken Maustaste doppelt auf den Button klicken. Es öffnet sich sofort ein so genanntes *Quelltextfenster*, in dem sich bereits drei, manchmal vier Programmzeilen befinden:



Die `procedure`-Zeile und die Zeile mit `begin` bilden die beiden *Kopfzeilen* des Rahmens. Die Zeile mit `end;` mit dem Semikolon bildet die *Fußzeile* des Rahmens. Das letzte `end.` mit dem Punkt hat mit dem Rahmen der Ereignisprozedur nichts zu tun – es darf aber *niemals gelöscht* werden.

In der ersten Kopfzeile lesen wir rechts neben dem Punkt die Vokabel `Button1Click` – damit erkennen wir zur Kontrolle, dass wir tatsächlich den *Rahmen der Ereignisprozedur für das Ereignis Klick auf das Objekt mit dem Namen Button1* vor uns haben.

Der Zwischenraum zwischen `begin` und `end;` ist von uns mit dem *Inhalt der Ereignisprozedur* zu füllen.

Hier müssen wir *programmieren*, d. h. wir müssen die *passenden Befehle* in der *richtigen Reihenfolge* eintragen, wobei wir uns an die *Regeln der Sprache Delphi-Pascal* halten müssen:

```
procedure TForm1.Button1Click(Sender: TObject);           // 1. Kopfzeile
begin                                                       // 2. Kopfzeile
    Showmessage('Der Button wurde geklickt')               // Inhalt
end;                                                         // Fußzeile
```

Bevor wir mehr zu den ersten Pascal-Regeln erfahren, wollen wir uns das Ergebnis ansehen: Beim Klick auf den Button zur Laufzeit erscheint tatsächlich in der Mitte des Bildschirms das von uns verlangte Mitteilungsfenster (`ShowMessage`) mit dem programmierten Inhalt:



Der von uns programmierte *Inhalt der Ereignisprozedur* besteht erst einmal nur aus einem einzigen Befehl – keine Sorge, das wird sich bald ändern. Die zwei Schrägstriche machen den Rest der Zeile programmtechnisch unwirksam – dahinter kann man folglich Bemerkungen (*Kommentare*) eintragen.

Groß- und Kleinschreibung der Befehle sind in Pascal nicht wichtig, dieselbe Ereignisprozedur könnten wir auch so schreiben:

```
procedure TForm1.Button1Click(Sender: TObject)           // 1. Kopfzeile
begin
    showMessage('Der Button wurde geklickt')             // Inhalt
end;
```

Wir verlieren aber dabei sehr viel an Übersichtlichkeit; der auszugebende Text muss natürlich trotzdem den Regeln der deutschen Sprache genügen. Die Übersicht geht ebenso verloren, wenn wir darauf verzichten würden, eingerückt zu schreiben.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    ShowMessage('Der Button wurde geklickt')             // Inhalt
end;
```

Oder wenn man gar darauf verzichtet, die wichtigen *Rahmen-Schlüsselwörter* `begin` und `end;` allein auf je eine Zeile zu schreiben:

```
procedure TForm1.Button1Click(Sender: TObject);
begin ShowMessage('Der Button wurde geklickt') end;
```

In Delphi-Pascal ist das Semikolon stets ein *Trennzeichen*.

Da der Inhalt unserer Ereignisprozedur diesmal nur aus einem Befehl bestand, war es überhaupt nicht notwendig, ein Semikolon zu setzen. Wir werden in weiteren Kapiteln sowieso noch genug Ärger mit diesem Semikolon bekommen.

Der Befehl `ShowMessage('Der Button wurde geklickt')` stellt in Wirklichkeit den *Aufruf der Prozedur* `ShowMessage` dar. Dabei wird der Text, der in dem Mitteilungsfenster erscheinen soll, in *einfache Hochkommas* ' ' gesetzt (auf der Tastatur meist gemeinsam mit dem Zeichen # zu finden).

2.2.2 Textfenster

Wichtiger Hinweis: Wer das folgende Delphi-Projekt nicht selbst entwickeln möchte, kann auf der Seite www.w-g-m.de/delphi.htm unter *Dateien für Kapitel 2* die Datei `DKap02.zip` herunterladen, die die Projektdatei `proj_222.dpr` enthält.

Rasch wird ein *Textfenster* auf dem Formular platziert (es bekommt von Delphi automatisch den Namen `Edit1`), und mit seinem Objektinspektor wird in der Zeile `Text` für den Start-Inhalt eine Zeichenfolge, zum Beispiel `Uenglingen`, vorbereitet:



Dann wird mit der linken Maustaste doppelt auf das Textfenster geklickt. Schon erhalten wir den *Rahmen für die Ereignisprozedur zum Standard-Ereignis beim Textfenster*. Welches Ereignis wird es wohl sein?

Der obersten Zeile des „geschenkten“ Rahmens können wir rechts vom Punkt die Vokabel `Edit1Change` ablesen. Das heißt, dass die Programmierer von Delphi der Meinung waren, dass eine *ändernde Nutzereinwirkung*, d.h. das Ereignis `Change` (Änderung) bei einem Textfenster, wohl am häufigsten der Ausgangspunkt für eine Reaktion sein wird. Nun brauchen wir nur noch den einen Informations-Befehl für den Inhalt zu schreiben:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
    ShowMessage('Es wurde geändert')           // Inhalt
end;
```

Lassen wir ausführen, so stellen wir tatsächlich fest, dass das einfache Hineinklicken in die Textbox noch nicht zur Ausführung dieser Ereignisprozedur führt. Das Ereignis *Klick* ist eben ein anderes Ereignis als *Änderung*.

2.2.3 Checkbox

Preisfrage: Welche Nutzereinwirkung wird wohl bei dem Bedienelement *Checkbox* am häufigsten auftreten und oft zu einer Reaktion Anlass geben?

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
    ShowMessage('Es wurde geklickt')           // Inhalt
end;
```

Natürlich – in der ersten Kopfzeile ist es ablesbar: Das *Standardereignis beim Bedienelement Checkbox* ist der *einfache Klick*. Wobei dieses Ereignis natürlich sowohl eintritt, wenn der Nutzer den Haken in der Checkbox *setzt*, als auch, wenn er den Haken *wegnimmt*. Klick ist Klick.

2.2.4 Scrollbar

Für das *Bedienelement Scrollbar* besteht das *Standard-Ereignis* in der *Änderung der Position des Reglers*. Der Doppelklick auf dieses Bedienelement im Entwurfsmodus liefert sofort den *Rahmen für eine Ereignisprozedur*, mit der man auf diese Nutzereinwirkung reagieren könnte:

```
procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
    ShowMessage('Es wurde am Regler geschoben') // Inhalt
end;
```

2.2.5 Radiobutton

Für ein *Bedienelement Radiobutton*, das wohl niemals allein auftreten wird, sondern immer mit anderen zusammen in einer *Gruppe* (GroupBox), ist ebenfalls der *einfache Klick* das *Standard-Ereignis*.

Das erfährt man aus dem zugehörigen Rahmen der Ereignisprozedur:

```
procedure TForm1.RadioButton1Click(Sender: TObject);
begin
    ShowMessage('Es wurde geklickt') // Inhalt
end;
```

2.2.6 Label

Welche Antwort sollten wir geben, wenn wir nach dem *Standard-Ereignis* zum *Bedienelement Label* gefragt werden? Oder – anders formuliert – welche Nutzerhandlung ist an einem Label am wahrscheinlichsten und sollte zu einer Reaktion führen?

Ein *Label* wird eigentlich nur *passiv* benutzt, um etwas mitzuteilen. Folglich lautet die spontane Antwort: Keine. Welcher Nutzer klickt schon ein Label an? Warum sollte er das tun?

Trotzdem haben die Entwickler von Delphi auch für ein Label ein *Standard-Ereignis* festgelegt. Lassen wir uns überraschen: `Label1Click` steht in der Kopfzeile des *Rahmens für die Ereignisprozedur*. Das beschreibt das Label-Standard-Ereignis. Nehmen wir es zur Kenntnis.

Sollten wir einmal aus irgendeinem Grunde für den Klick eines Nutzers auf ein Label eine Ereignisprozedur zu schreiben haben, wissen wir folglich, dass wir damit das Standard-Ereignis am Label behandeln.

2.2.7 Formular

Wenn wir im Entwurf auf das *leere Formular* doppelt klicken erhalten wir sofort folgenden *Ereignisprozedur-Rahmen*:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    // Inhalt
end;
```

Das bedeutet, dass bei dem grundlegenden visuellen Objekt, dem Formular, das *Erzeugen* (Create) das *Standardereignis* ist. Hier haben die Programmierer von Delphi keine direkte Nutzereinwirkung auf das Formular zum Standard-Ereignis erklärt. Stattdessen kann man als *Inhalt der Ereignisprozedur* alle Befehle hineinschreiben, die automatisch ausgeführt werden, wenn das *Formular hergestellt* wird. Wir werden davon bald, nämlich im Abschn. 4.7.5, Gebrauch machen.

2.3 Einfache Ereignisprozeduren zu Nicht-Standard-Ereignissen

Wie erfährt man überhaupt, für welche Ereignisse man Inhalte von Ereignisprozeduren programmieren kann?

Mit anderen Worten: Für welche Nutzereinwirkungen auf ein Bedienelement ist in Delphi eine Reaktion programmierbar?

Bisher wissen wir doch nur, wie wir eine *Reaktion beim jeweiligen Standard-Ereignis* erzeugen können.

2.3.1 Reaktionen auf Mausbewegungen

Was ist zu tun, wenn wir beispielsweise eine Reaktion programmieren wollen, die stattfindet, wenn der Nutzer die Maus lediglich *über einen Button bewegt*?

Zwei Fragen müssen dazu beantwortet werden:

- Erste Frage: Gibt es überhaupt die Möglichkeit in Delphi, zum Ereignis *Mausbewegung über einem Button* eine Ereignisprozedur schreiben zu können?
- Zweite Frage: Wie erhält man dann den *Rahmen für die Ereignisprozedur*?

Beide Antworten sind nicht schwierig zu erhalten: Nachdem wir einen Button auf dem Formular platziert haben (zum Beispiel mit der Beschriftung NORD), wählen wir im Objektinspektor das rechte Registerblatt mit der Überschrift EREIGNISSE aus. Und da ist sie schon, die *Liste aller in Delphi behandelbaren Button-Ereignisse*, d. h. die *Liste aller Arten von Nutzereinwirkungen am Button*, für die wir in Delphi Reaktionen programmieren können. Über die rechte Maustaste lässt sich die Anordnung entweder alphabetisch oder nach Kategorien geordnet anzeigen (siehe Abb. 2.6).

Und in dieser Abbildung können wir erkennen: Weit *mehr als ein Dutzend Ereignisse am Button* sind behandelbar; für sie könnten wir folglich Ereignisprozeduren vorbereiten.

Eines davon ist das bekannte Standard-Ereignis Klick, das sich natürlich auch in der Liste befindet. Dazu gibt es also beim *visuellen Objekt Button* eine Fülle behandelbarer *Nicht-Standard-Ereignisse*.

Wohlgemerkt: Diese Aussage bezieht sich auf das aktuelle Delphi, in anderen Entwicklungssystemen mit ähnlichem Aufbau kann es durchaus weniger oder mehr behandelbare

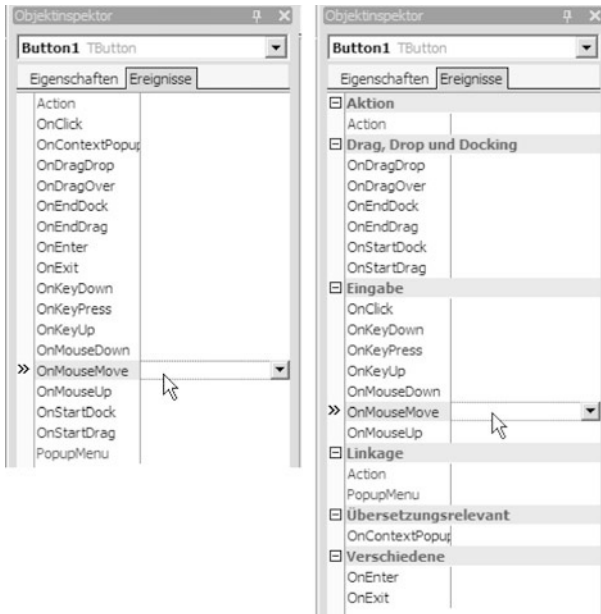


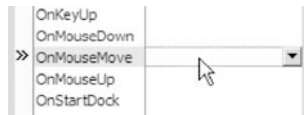
Abb. 2.6 Objektinspektor des Buttons, Registerblatt EREIGNISSE

Ereignisse geben. Auch können künftige Delphi-Versionen durchaus mit längeren Listen versehen sein.

Jedes Ereignis beginnt mit den zwei Buchstaben `On` – ins Deutsche am besten übersetzbar mit *bei*.

In einer Zeile erkennen wir das Ereignis `OnMouseMove`. Zu deutsch: *Bei-Maus-Bewegung*. Da haben wir es doch, wonach wir suchen. Folglich ist die erste Frage beantwortet: Wir können durchaus auch zum *Ereignis Mausbewegung über dem Button* eine Ereignisprozedur schreiben.

Dazu wählen wir in der Liste aller behandelbaren Ereignisse das Ereignis `OnMouseMove` aus. Dann öffnet sich rechts ein leeres weißes Feld:



Wichtiger Hinweis: Wer das folgende Delphi-Projekt nicht selbst entwickeln möchte, kann auf der Seite www.w-g-m.de/delphi.htm unter *Dateien für Kapitel 2* die Datei `DKap02.zip` herunterladen, die die Projektdatei `proj_231.dpr` enthält.

Weiter geht es schließlich mit einem schnellen Doppelklick in dieses leere weiße Feld – und schon öffnet sich wieder wie bei den Standard-Ereignissen das *Quelltextfenster mit dem Rahmen der Ereignisprozedur*. Damit ist auch die zweite Frage beantwortet.

In der obersten Kopfzeile des *Rahmens der Ereignisprozedur* liest man dann rechts vom Punkt die Zeichenfolge `Button1MouseMove`, es handelt sich tatsächlich um den Rahmen

für die Ereignisprozedur zum *Ereignis Mausbewegung über dem Button*. Für den *Inhalt der Ereignisprozedur* wollen wir vorerst wieder nur eine einfache Mitteilung wählen.

```
procedure TForm1.Button1MouseMove (
    Sender:TObject;
    Shift:TShiftState;
    X,Y:Integer
);
begin
    ShowMessage('Der Mauszeiger bewegt sich über dem Button NORD')
    // Inhalt
end;
```

Dabei ist die lange Kopfzeile hier aus schreibtechnischen Gründen zerlegt worden; im eigentlichen Delphi-Quelltextfenster ist das nicht nötig.

Zur Übung könnten *weitere drei Buttons* mit den Beschriftungen OST, WEST und SÜD an den entsprechenden Stellen, etwas entfernt voneinander, auf dem Formular platziert werden. Zu jedem Button wird dann für das Ereignis Mausbewegung der Rahmen der zugehörigen Ereignisprozedur beschafft, und anschließend wird jeweils die passende Mitteilung programmiert.

Was ist aber, wenn sich der Mauszeiger *nicht über einem der Buttons* befindet? Dann – logisch – befindet er sich über dem *Formular*. Die Liste der Formular-Ereignisse, zu denen eine Ereignisprozedur geschrieben werden kann, ist lang, sehr lang. Mehr als dreißig Einträge umfasst sie; natürlich befindet sich auch der Eintrag *OnMouseMove* in dieser Liste. Wer jedoch auf die logische Idee kommt, folglich eine fünfte Ereignisprozedur zum Ereignis *Mausbewegung über dem Formular* zu programmieren, der wird einstweilen viel Ärger bekommen.

Warum?

Nun, das Mitteilungsfenster, das mit dem Aufruf der Prozedur *ShowMessage* angefordert wird, wird von Delphi stets in der Mitte des Bildschirms geöffnet, also über dem Formular. Wenn man diese Mitteilung mit OK bestätigt und das Mitteilungsfenster verschwindet – wo befindet sich dann der Mauszeiger? Natürlich – wieder über dem Formular. Das Mitteilungsfenster erscheint unverzüglich wieder und so weiter. Man kommt aus dem Teufelskreis nur heraus, wenn das Mitteilungsfenster vor dem Schließen vorsichtig ganz an den oberen Rand des Bildschirms gezogen wird. Ein schöner Effekt, aus dem man viel lernen kann.

Für die beiden anderen behandelbaren Mausereignisse mit den Bezeichnungen *OnMouseDown* und *OnMouseUp* könnten wir ebenso interessante Reaktionen programmieren.

Übersetzen wir die beiden englischen Vokabeln in die deutsche Sprache:

- *OnMouseDown* = Beim *Runterdrücken der linken Maustaste*, während sich der Mauszeiger über dem Bedienelement oder dem Formular befindet.
- *OnMouseUp* = Beim *Loslassen der linken Maustaste*, während sich der Mauszeiger über dem Bedienelement oder dem Formular befindet.

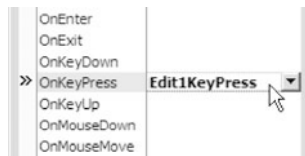
2.3.2 Reaktionen auf Tastendruck

Wichtiger Hinweis: Wer das folgende Delphi-Projekt nicht selbst entwickeln möchte, kann auf der Seite www.w-g-m.de/delphi.htm unter *Dateien für Kapitel 2* die Datei `DKap02.zip` herunterladen, die die Projektdatei `proj_232.dpr` enthält.

Betrachten wir ein *Formular* mit einem darauf befindlichen *Textfenster* (Edit), in das der Nutzer selbst etwas eintragen kann oder in dem er eine vorhandene Eintragung verändern kann. Das *Ereignis Änderung* ist das *Standard-Ereignis* am Bedienelement Textfenster, wir haben es im Abschn. 2.2.2 schon kennen gelernt.

Wir stellen uns die Frage: Für wie viele und welche *Nicht-Standard-Ereignisse am Textfenster* könnten wir Reaktionen programmieren?

In der Registerkarte EREIGNISSE sind wiederum viele Einträge enthalten. Für die Nutzereinwirkung *Taste wurde gedrückt*, auf englisch `OnKeyPress`, ist bereits der *Rahmen der Ereignisprozedur* angefordert worden – erkennbar an dem Eintrag in dem weißen Feld neben `OnKeyPress`:



Der *Inhalt der Ereignisprozedur* besteht wieder vorerst nur aus dem einfachen Aufruf der Prozedur `ShowMessage`, mit dem der Nutzer über das Eintreten des Ereignisses informiert wird.

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    ShowMessage('Eine Taste wurde gedrückt')
end;
```

Wer die Ereignisprozedur testet, wird erstaunt feststellen, dass keinesfalls jeder Tastendruck die programmierte Reaktion hervorruft. Weder auf die *Shift*-Taste noch auf die Taste *Strg* noch auf die Funktionstasten F1 bis F12 gibt es eine Reaktion.

Ist das schlecht? Sicher nicht, denn sonst könnte jeder Programmierer mittels einer Ereignisprozedur diese wichtigen Tasten, die für die *Arbeit des Betriebssystems* unverzichtbar sind, einfach außer Betrieb nehmen oder in ihrer Wirkung verfälschen.

Für die beiden anderen Tastatur-Ereignisse

- `OnKeyDown` = *Bei heruntergedrückter Taste,*
- `OnKeyUp` = *Bei losgelassener Taste,*

werden wir später vielleicht interessante Anwendungen finden; ihre Nutzung im Zusammenhang mit der Prozedur `ShowMessage` ist nicht günstig.

Abb. 2.7 Der Button oben links besitzt den Fokus



Abb. 2.8 Das Textfenster besitzt den Fokus



2.3.3 Fokus-Ereignisse

Wichtiger Hinweis: Wer das folgende Delphi-Projekt nicht selbst entwickeln möchte, kann auf der Seite www.w-g-m.de/delphi.htm unter *Dateien für Kapitel 2* die Datei `DKap02.zip` herunterladen, die die Projektdatei `proj_233.dpr` enthält.

Bevor wir uns den beiden *Nicht-Standard-Ereignissen* `OnEnter` und `OnExit` zuwenden, die für jedes Bedienelement behandelbar sind, müssen wir erst einmal den wichtigen Begriff des *Fokus* klären.

Zur Illustration platzieren wir auf einem Formular fünf Bedienelemente, und wählen dabei, um später den Wechsel des Fokus gut beobachten zu können, eine ganz bestimmte Reihenfolge: Zuerst kommt nach links oben die Schaltfläche `Button1`. Rechts daneben wird das Textfenster `Edit1` angeordnet. Erneuter Griff in die Liste der Bedienelemente, und links unten wird weiter die waagerechte `Scrollbar1` angeordnet. Rechts unten wird danach der `Button2` platziert, und zum Schluss ordnen wir rechts oben die `CheckBox1` ein. Die Entwurfsphase ist beendet, wir speichern `Unit` und `Projekt` nacheinander so, wie es in Abschn. 1.5 beschrieben wurde.

Ein Klick auf die Schaltfläche mit dem nach rechts gerichteten Dreieck oder ein Druck auf die Taste `F9` – die *Laufzeit* beginnt. Sehen wir in Abb. 2.7 genau hin: Um die Beschriftung von `Button1` zieht sich eine gestrichelte Linie. Das bedeutet: Der erste Button *besitzt den Fokus*.

Nun sollten wir die Maus weglegen und die *Tabulator-Taste* auf der Tastatur suchen. Sie befindet sich ganz links und ist beschriftet mit zwei übereinander liegenden, entgegengesetzten Pfeilen.

Abb. 2.8 zeigt uns, was sich *nach Druck auf die Tabulatortaste* ändert: Die Linie um die Beschriftung von `Button1` ist verschwunden, dafür ist der Inhalt von `Edit1` blau unterlegt. Die Unterschrift von Abb. 2.8 erklärt es: Der *Fokus*, d. h. die Markierung desjenigen Bedienelements, das *aktuell bedienbar* ist und Nutzereingaben entgegennehmen kann, ist zur *Textbox* gewandert.

Wohin wandert der *Fokus* beim nächsten Druck auf die Tabulatortaste? Wird er oben weiterwandern oder wird ein anderes Bedienelement fokussiert? In Abb. 2.9 sehen wir



Abb. 2.9 Die Scrollbar ist nun fokussiert

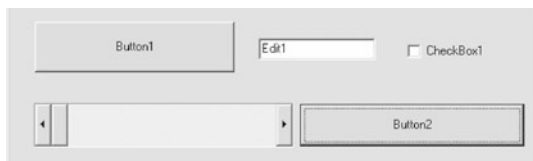


Abb. 2.10 Der Button unten rechts hat den Fokus



Abb. 2.11 Fokussierte Checkbox

die Antwort: Die *Scrollbar* bekommt den Fokus. Warum? Erinnern wir uns an die Reihenfolge, in der wir die Bedienelemente auf dem Formular platzierten. Hier zeigt sie sich.

Was kommt als Nächstes? Erinnern wir uns – danach hatten wir den *zweiten Button* auf das Formular gezogen. Erneuter Druck auf die Tabulatortaste fokussiert folglich den zweiten Button rechts unten (siehe Abb. 2.10).

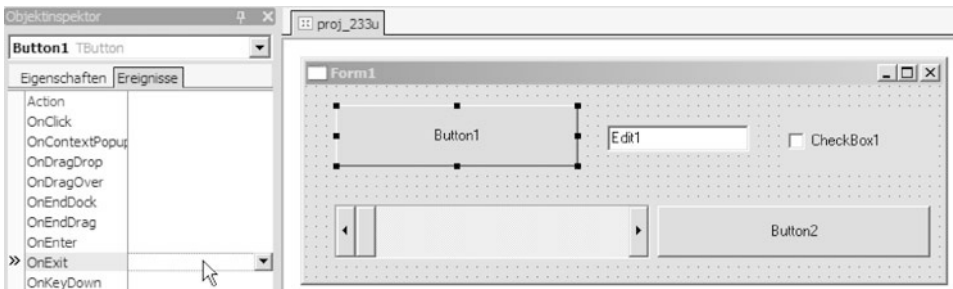
Als letztes hatten wir die Checkbox angeordnet, demnach erhält sie, wie Abb. 2.11 zeigt, erst beim letzten Druck auf die Tabulatortaste den *Fokus*. Wer es ausprobier, der stellt natürlich sofort fest, dass bei erneutem Druck auf die Tabulatortaste der *Fokus wieder reihum* wandert.

Wenn der Nutzer den Fokus auf ein Bedienelement setzt, will er es anschließend bedienen.

Am deutlichsten wird das bei einem Textfenster. Erst wird fokussiert – mit Tabulatortaste oder Maus wird editiert, eingetragen oder geändert oder gelöscht. Arbeitet der Nutzer mit der *Maus*, dann verbindet er im Regelfall das *Setzen des Fokus* sofort mit der Bedienung: Ein *Mausklick auf einen Button* setzt den Fokus auf diesen Button, gleichzeitig erfolgt bereits der Klick. Würde dagegen der Button nicht mit der Maus, sondern mit der Tabulator-Taste fokussiert, dann würde erst die Leertaste oder Enter-Taste den Klick auslösen.

Zu jedem Bedienelement kann festgestellt werden, ob ein Nutzer den Fokus auf dieses Bedienelement setzt. Dieses Ereignis heißt bei Delphi `OnEnter` – es ist offensichtlich unter der *Liste der Nicht-Standard-Ereignisse* zu suchen.

Lassen wir uns zum Beispiel mittels einer Ereignisprozedur mitteilen, wenn der erste Button mit dem Namen `Button1` den Fokus erhält. Was ist zu tun? Im Entwurfsmodus wird `Button1` und in der Registerkarte `EREIGNISSE` des Objektspektors wird das Ereignis `OnEnter` ausgewählt:



Ein Doppelklick in das leere weiße Feld, und schon erhalten wir den *Rahmen* für die entsprechende Ereignisprozedur, den wir wie bisher nur mit einem Aufruf der Prozedur `ShowMessage` füllen:

```
procedure TForm1.Button1Enter(Sender: TObject);
begin
    ShowMessage('Button1 hat den Fokus')
end;
```

In gleicher Weise können wir uns natürlich mit einer *Ereignisprozedur zum Ereignis* `OnExit` auch sagen lassen, wenn ein Bedienelement *den Fokus verliert*. Sehen wir uns beispielsweise auch dazu die Ereignisprozedur an.

```
procedure TForm1.Button2Exit(Sender: TObject);
begin
    ShowMessage('Button 2 verlor den Fokus')
end;
```

Später, im Abschn. 4.4, werden wir derartige Ereignisprozeduren nutzen, um den Nutzer beim Versuch zu ertappen, eine Textbox zu verlassen, ohne dass er darin die gewünschte Angabe eingetragen hat. Denn beim Wechsel zu einem anderen Bedienelement würde gerade das *Ereignis Fokusverlust des Textfensters* eintreten – und das können wir nun behandeln. Wir werden dann den Nutzer zum korrekten Eintrag zwingen, indem wir ihm eine entsprechende Mitteilung zukommen lassen und dann den Fokus zurücksetzen.

Kommen wir zum Schluss dieses Abschnitts noch einmal zur *Reihenfolge der Fokussierung* zurück.

Diese Reihenfolge ist deshalb so bedeutsam, weil es immer noch und immer wieder Nutzer gibt, die mit Hilfe der Tabulatortaste von Bedienelement zu Bedienelement wechseln, vor allem dann, wenn es sich um Textfenster handelt, in die etwas einzutragen ist oder in denen Einträge zu verändern sind. Denn diese Art der Bedienung geht wesentlich schneller als der Griff nach der Maus und das Suchen mit dem Mauszeiger.

Offenbar werden die Bedienelemente aber von Delphi so fokussiert, wie sie anfangs auf dem Formular platziert wurden.

Das bedeutet, dass das zuletzt eingefügte Bedienelement auch zuletzt fokussiert wird, selbst wenn es oben links stehen sollte. Mit BEARBEITEN → TABULATORREIHENFOLGE lässt sich das jedoch problemlos ändern:



Grundkurs Programmieren mit Delphi
Systematisch programmieren lernen für Einsteiger
Matthäus, W.-G.
2016, XIV, 309 S. 280 Abb., Softcover
ISBN: 978-3-658-14273-5